Shared Memory Architecture and Explored Alternatives for Interoperability
Technical Session: Collaborative Simulation and Testing

Dr.rer.nat. Gary E. Lohman
Naval Air Systems Command (NAVAIR)
Integrated Battlespace Simulation and Test (IBST) Dept.
Air Combat Environment Test & Evaluation Facility (ACETEF)
48150 Shaw Rd. Bldg 2109, S115, Patuxent River, MD 20670
301-757-1161 (Phone), 301-342-6381 (FAX)
gary.lohman@navy.mil

David W. Mutschler, Ph.D.
Naval Air Systems Command (NAVAIR)
Integrated Battlespace Simulation and Test (IBST) Dept.
Air Combat Environment Test & Evaluation Facility (ACETEF)
22367 Cedar Point Rd. Bldg 2185, Rm. 2260-B4, Patuxent River, MD 20670
301-342-6837 (Phone), 301-342-2366 (FAX)
david.mutschler@navy.mil

1 Interoperability and ACETEF

The Air Combat Environment Test & Evaluation Facility (ACETEF) is a major component of the Naval Air Systems Command (NAVAIR) Integrated Battlespace Simulation and Test (IBST) Department. The IBST Department is a collection of geographically distributed yet integrated test facilities and many diverse activities all designed to provide effective, affordable and repeatable test & evaluation (T&E) capabilities for a variety of naval aircraft, weapon systems, and other platforms. IBST includes facilities in both the Naval Air Warfare Center Weapons Division (NAWCWD) and the Naval Air Warfare Center Aircraft Division (NAWCAD). Major Installed Systems Test Facilities (ISTFs) within NAWCWD include the Radar Reflectivity Laboratory (RRL) in Pt. Mugu, CA, as well as the Integrated Battlespace Arena (IBAR) and the Missile Engagement Simulation Arena (MESA) in China Lake, CA. Major test facilities within NAWCAD include the Surface/Aviation Interoperability Laboratory (SAIL), the Manned Flight Simulator (MFS), and the Air Combat Environment Test & Evaluation Facility (ACETEF) at Patuxent River, MD. Other activities within IBST include Electromagnetic Environmental Effects (E3) facilities, NAVAIR High Performance Computing (HPC) Centers, and the NAVAIR Research, Development, Test, and Evaluation (RDT&E) network domain [YM07].

The ACETEF concept reaches back to the 1970's with the convergence of the F/A-18 Hornet, the AV-8B Harrier, and the SH-60B Sea Hawk programs at the Naval Air Test Center, as it was then known. The first operations paradigm was "Fly-Analyze-Fix." By the early 1990's this operations paradigm had shifted to "Simulate-Stimulate-Analyze & Fix-Fly" [OCD97]. A key enabler in this paradigm is simulation, which is the "goal-directed experimentation with dynamic models, i.e., models with time-dependent behavior" [Ore02]. ACETEF extends the simulation concept to encompass a "coherent

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **2007**	2. REPORT TYPE	3. DATES COVERED **00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE **Shared Memory Architecture and Explored Alternatives for Interoperability Technical Session: Collaborative Simulation and Testing**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Air Systems Command (NAVAIR,Integrated Battlespace Simulation and Test (IBST) Dept.,48150 Shaw Rd. Bldg 2109, S115,Patuxent River,MD,20670**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S)
	11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
ITEA 2007 Modeling & Simulation Conference, Las Cruces, NM, Dec. 10?13, 2007

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**35**	

environment” in which other dynamic functional components ranging from man-and equipment-in-the-loop to simulation and stimulation assets can be “immersed” and interact. In this integrated environment of immersed dynamic systems, emergent effects can be observed and analyzed so that the whole of ACTEF can actually be greater than the sum of its parts. At a physical level, ACETEF is a collection of laboratories and facilities that includes two anechoic chambers (the Advanced System Integration Laboratory (ASIL) and the smaller Aircraft Anechoic Test Facility (AATF)), the Warfare Simulation Lab, the Threat Air Defense Lab (TADL), the Communication, Navigation, and Identification (CNI) lab, the Electronic Warfare Integrated Systems Test Laboratory (EWISTL), and the Electro Optic Infrared (EOIR) Laboratory. While these laboratories maintain their ability to operate independently, at the deeper logical level their specific shared interoperable architecture facilitates integration in a tightly-coupled real-time manner. The Manned Flight Simulator (MFS) is also able to interact with ACETEF in this real-time, integrated operational mode.

Since ACETEF’s inception, the evolution of modern warfighting doctrine has put increasing operational importance and even dependence on complex interactions. The emergent effects that were side-effects in the past are rapidly becoming the mainstays of “edge entities,” capable of conducting highly responsive military missions [Alb03]. The ACETEF concept in supporting such highly complex integrated T & E activities is more relevant today than ever. At the heart of the ACTEF concept is a particular form of interoperability that engenders coherency of interactions, consistency of data modeling and synchronization of distributed actions. Whereas IEEE defines interoperability broadly as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” [IEEE90], ACETEF’s interoperability is based on concurrency. Concurrency is a property of systems in which several computational processes are executing at the same time, and interacting with each other [ROS97].

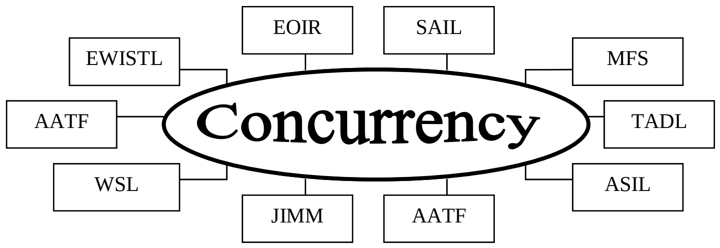


Figure 1 – ACETEF Functional Basis.

The ACETEF functional basis as depicted in Figure 1, provides the motivational backdrop for the ensuing discussion regarding shared memory and alternatives from the perspective of leveraging different concurrency models.

2 Concurrency – A closer Look

Concurrency is one of those quintessential challenges of interoperation with respect to information processing systems. Concurrency applies to all levels of interaction within a computing environment from hardware and the operating system to the functional applications, which is the level of interest with respect to ACETEF interoperability.

For simplicity sake, application concurrency can be taken as the execution of multiple interacting computational tasks implemented as separate processes. The functional perspective of application-level concurrency involves the implementation of a concurrency model that includes an inter-process communication (IPC) method with a coherence protocol, a data consistency model and a synchronization mechanism.

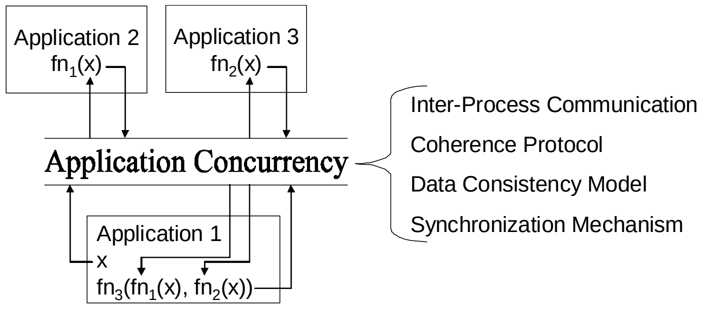


Figure 2 – Motivating the challenge of application concurrency - application1 provides the seed value x for fn1 and fn2 that can be executed more-or-less in parallel in applications 2 and 3 respectively. Application 1 relies on these functional results as the arguments of its own function.

The simple situation depicted in Figure 2 already illustrates the need for some degree of coherence, consistency and synchronization at the level of the concurrent applications' interacting functions. There must also be an understanding atomicity across the applications, namely those actions that are indivisible, because the concurrency model must address these core issues at the appropriate atomic level. From the ACETEF perspective, atomicity can become a highly complex issue on its own merit. The individual models behind the simulations and stimulators as well as the dynamic interactions with human interfaces and instrumentation can exist at different levels of detail. The individual levels of detail are related in the sense of physical causality, such that patterns of conjoined or sequential events at one level will reflect events at a higher level of abstraction. In a practical sense, if the functional components do not share a common level of detail, then the atomic abstraction will impact the coherence, consistency and synchronization details of the concurrency model.

Concurrency is more than just a data protocol. Data-level protocols like Distributed Interactive Simulation (DIS) put the primary focus on the data formatting and passing aspect of sharing and not the concurrent interaction of processes actually sharing the information. This provides a lowest-common-denominator basis for data consistency, but does little with respect to coherence and synchronization. By contrast the High Level Architecture (HLA) protocol facilitates data sharing along the lines of a Federate Object Model (FOM), which enables data sharing within a commonly agreed upon data-context. This higher-level data consistency model adds some coherence and synchronization potential. The Test and Training Enabling Architecture (TENA), which is based on the Common Object Request Broker Architecture (CORBA) concept, goes beyond the mere data passing aspect to include the methods and marshalling instructions necessary for concurrently mediating process interactions involving the data at a functional level. These three common modeling and simulation (M&S) data protocols represent vastly different forms of interoperability from a concurrency point of view.

Concurrency is also more than just the choice of IPC. Although IPC methods come in many different technical and OS-specific forms, in an abstract application functional sense it boils down to a "sharing" of something such as a *file*, a block in *memory*, a *connection* for passing messages, or an *environment* for passing signals or control instructions. Consequently, two classes of explicit inter-process communications can be readily identified. On one hand, there is the file and memory sharing, which typically involves some kind of locking-based protocol as in the shared-memory concurrency model. On the other hand, passing messages, signals and instructions through a shared connection or environment typically involves some kind of process calculi or actor concurrency model. In thread discussions, these two classes are often succinctly described as "variable sharing" or "message passing."

In its original form, shared memory was both an efficient IPC method and a concurrency model. As an IPC method, once the memory is mapped into the address space of the process sharing the memory region, memory management functions aside, the data is efficiently passed between processes without executing data calls through the kernel. Today, the use of shared memory as an IPC may be disguised under many different application programming interfaces (API), but most modern implementations on Windows, UNIX and Linux platforms actually employ some kind of file memory mapping as the actual explicit inter-communication mechanism. As a concurrency model, shared memory involves a straightforward coherence protocol of the form: locking memory; manipulating shared memory followed by freeing memory. Consistency and synchronization are then maintained by simply imposing basic rules governing the requesting and order of shared memory accesses by the individual processes. While the details of the coherence protocol and consistency model change with memory mapped files, the basic "locking" effect at the shared variable level is still preserved behind the API, so that the coherence model is essentially the same in outward appearance.

While the shared memory concurrency model is simple and in many respects elegant to the point of being virtually transparent to the user, this also speaks to the level of atomicity involved. For example, if two applications have functionally identical variables, then those variables can be efficiently "shared" through the shared memory concurrency model quite effectively. Now consider the danger in shared memory if a process crashes while manipulating the "critical region" of memory, or if it locks too frequently. In the case of fine-grain sharing, consider the effect of a "deadlock" where two processes hold "hostage" the portion of memory that each other is waiting for. Such low-level "exceptions" might actually be easily resolvable at a higher task-level of understanding. Unfortunately, at the atomicity of the concurrency model employed, these situations may be inherently ambiguous. If the atomicity of the process interaction is at a higher functional level so as to include data structures, control methods and even program objects, then the intrinsic support of a more complex concurrency model may be required.

Message passing IPCs fundamentally offer greater flexibility for supporting a broader class of concurrency models at both the variable-level and beyond. Furthermore, message passing IPCs provide an important basis for extending the concept of concurrency across processes running on multiple and possibly disparate compute platforms as in a networked environment. The proliferation of high-bandwidth standards-based networks combined with easy-to-program abstractions such as sockets, makes the message passing paradigm in distributed systems generally appealing, but we must keep in mind the fact that concurrency is more than the IPC chosen as the means of implementation.

As a point in case, consider Distributed Shared Memory (DSM). DSM implemented at the operating system level can be thought of as extensions of the underlying virtual memory architecture and as such is completely transparent to the applications, but is also particular to a small family of operating systems and generally require that all compute platforms be running the same OS across a specific network backbone. If the OS does not support DSM natively, which most do not, then the distributed shared memory concept must be created by (1) preserving locally the shared memory look and feel (i.e. appropriate API "disguise") and (2) enforcing globally the shared memory concurrency model. This is typically achieved by using some form of a message passing IPC. As a notional DSM implementation, consider a local process that would normally interact with another local process through shared memory. The process now interacts through local shared memory with a proxy that is the DSM local client. The DSM clients share a message-passing connection at the core of their infrastructure. The IPC implementation in this case supports a process algebra that extends the local shared memory concurrency model to the other clients (e.g. locking one; locks all) as well as mediates and synchronizes the data manipulations across the clients for a net reflective data effect. Whether the DSM implementation is software based using standard networking hardware and protocols like TCP/IP, or employs proprietary hardware and firmware to offload the process-algebra and communications processing, the effect of a shared memory concurrency model remains.

While shared memory represents the low-end of the concurrency model spectrum of complexity, the Common Object Request Broker Architecture (CORBA) concept represents a considerably higher degree of supportable concurrency complexity. Fundamentally based on a message passing paradigm across standard computer networks, CORBA is a formal standard defined by the Object Management Group (OMG) for enabling software components written in various computer languages and running on multiple computers to operate concurrently. CORBA capitalizes on the object oriented (OO) nature of most modern languages and their use of objects in order to enable interaction through the sharing of objects. Objects embody not only the data but also the encapsulation of methods for manipulating the data in a functionally consistent context. In particular, CORBA uses an interface definition language (IDL) to define the objects and services in a language-independent manner. When compiled using the IDL, a client side stub-code, and a server side skeleton-code are created (Figure 3).

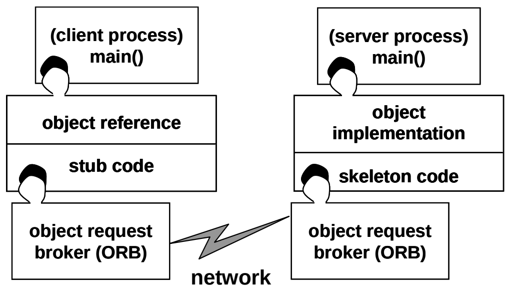


Figure 3 – The CORBA concept supports by design concurrency as well as interoperability.

When a client makes a call to a remote object, the stub provides the interface or proxy to the remote object along with the marshalling instructions for the client's object resource broker (ORB). The skeleton provides the server's ORB with the means to interpret the client's method and the un-marshalling instructions. On return response from the remote object, the roles are essentially reversed. Application level concurrency lies in the particular methods and marshalling instructions, which can be designed around various concurrency models. CORBA's growth as a concurrent programming middleware from the mid 1990's on was marked by sometimes even more rapid development and commercial acceptance of JAVA and Enterprise Java Beans (EJB). For the purpose of discussion this it is worth noting that even in 2007 these approaches continue to compete so that the only clear point may be simply that the final chapter of the quest for such high-end flexible, language and platform independent concurrent middleware is yet to be written.

3 Concurrency Evolution at ACETEF

In the early 1990's when ACETEF was architecting its integrated, concurrent operating capabilities, comparatively slow processors, low bandwidths and severely limited dynamic memory narrowed the concurrent processing options to shared memory, both locally and distributed using proprietary hardware. Using the simulation concept for the concurrent environment basis, the ACETEF architects took a warfare model of the same family as the TAC Suppressor and made some specific, yet very profound alterations. The model's game engine was based on a semantic network. Such engines work on the principle that any event, like the movement of a game entity, changes certain parts of the network and triggers a predicate logic traversal of the network to determine the response of the entity and all other entities, which in turn gets reflected back in the network in preparation for the next event. The significance of this kind of logic-engine architecture is that if certain parts of the semantic network were made externally accessible, then the behaviors in addition to the triggering events could be driven externally. ACETEF's key alteration of the model was in fact to make parts of the network externally accessible through a shared memory interface as well as add simulation control and contingency behaviors [Lat95]. By providing for its own memory management through the model's control, the interface could serve as a coordination protocol for the functionally organized memory blocks. This interface-protocol became known as Simulated Warfare Environment Data Transfer (SWEDAT). Certain SWEDAT shared memory blocks correspond to entity position and orientation (both actual and perceived), emissions, scenario configuration data, timing and coordination. Other blocks, however, correspond to low-level model-specific mailboxes where battlespace interactions with entities as well as entity behaviors can be shared. In this way, data can be shared at the "variable-sharing" level within a shared memory concurrency model, while actual interactions take place within the model at higher battlespace-relevant levels of atomicity with the effects reflected back to the lower SWEDAT level. The net effect is a stimulus-decision-response (SDR) interaction between entities within the model and their outside world counterparts (Figure 4).

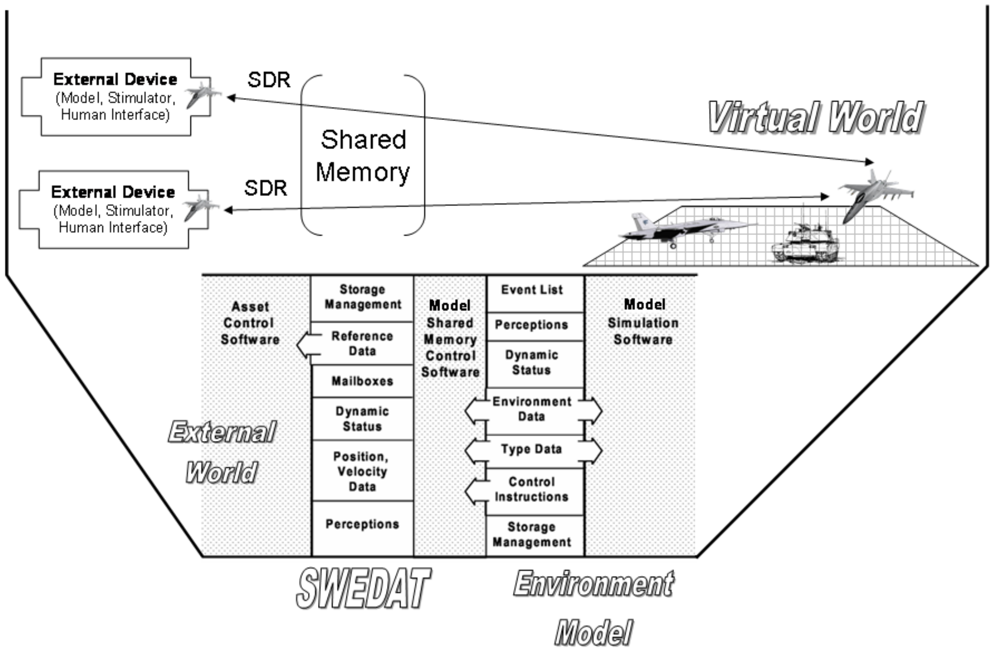


Figure 4 – The shared memory based SWEDAT interaction paradigm allows the model to act as a virtual world and mediate a stimulus-decision-response concurrent connection between external devices and their virtual counterparts.

This creates a profound concurrent computing capability by enabling the model to function as a kind of "virtual world hosting entities" whose behaviors could be concurrently driven by external devices such as models, stimulators and man-machine interfaces. Since each external entity within this construct "sees" the "reflection" of the entity in the virtual world, including each other's effects without explicitly "knowing" about each other, a highly effective agent-based integration paradigm for concurrently operating external devices is achieved. While the low level "variable-sharing" data atomicity through SWEDAT can demand considerable effort with a steep learning curve for creating, debugging, operating and maintaining external interfaces, the effective SDR interaction addresses concurrency at the higher-level of battlespace interactions, which defines the atomicity of the overall concurrency model in effect.

The SWEDAT protocol continues to be very important for ACETEF integrated operations, and it is still based on shared memory (including DSM). The descendant of the originally modified model, known today as the Joint Integrated Mission Model (JIMM), continues to control those blocks of the shared memory that are specific to external, concurrent control [Lat07]. In the post "AI Winter," as some have called it, it is ironic to see the number of new agent-based and semantic network driven model concepts arising today, while the agent-based roots of successful models like JIMM have been virtually forgotten [LS07] In addition to supporting real-time integrated exercises involving direct stimulation of aircraft systems under test with emitted energy, SWEDAT can also link ACETEF into distributed exercises via interfaces using DIS, HLA, the TENA, and other methods (Figure 5). Furthermore, these methods can also operate simultaneously with each other or with laboratory interfaces as required. Multiple copies of JIMM can also operate directly through SWEDAT [Mut05]. In addition, this allows simulators such as the Joint Semi-Automated Forces (JSAF) simulator or the Enhanced Air Defense Simulator (EADSIM) to provide all or part of the threat environment during exercises [IBST07].

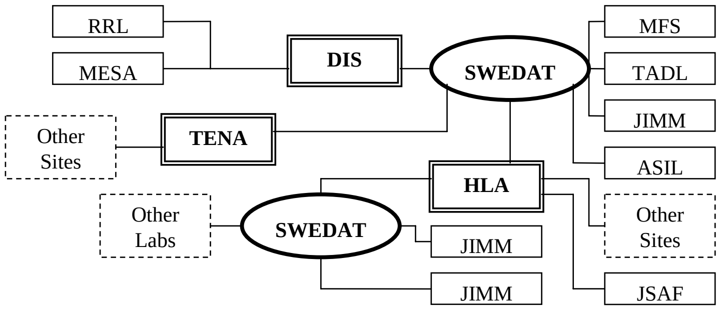


Figure 5 – Notional Integrated Exercise with Distributed Sites

Over the past few years, components of an interface library have evolved using an object-oriented construction to extend interactions beyond SWEDAT [MA06]. In this approach, methods employed to access SWEDAT data are kept in a derived class. However, actual SWEDAT interaction is maintained in a base class. Since much of the SWEDAT data does correspond to information found in protocols like DIS, HLA and TENA, this library can be extended to handle other protocols through a plug-in architecture that continues to evolve as depicted in Figure 6.

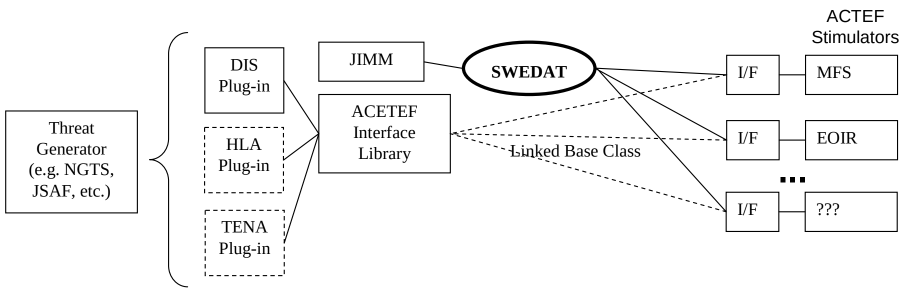


Figure 6 – Current state of the ACETEF architecture

The data interchange structure and management control still relies on SWEDAT under JIMM control. Two proposals have been submitted to fully separate SWEDAT control from JIMM [Mut03], [Mut04a]. While this would take SWEDAT and the shared memory management out of the JIMM model, the net effective SDR concurrency paradigm provided by using JIMM as a kind of virtual world would no longer be intrinsic. This means that data is being interchanged in the runtime environment, but the true concurrency burden, i.e. coherency, consistency and synchronization, gets left largely to the assumed concurrent operating capabilities of the interconnected components.

The SWEDAT and interface library are predicated locally on a shared memory data model so that the current implementation relies heavily on a costly distributed reflective shared memory architecture involving proprietary hardware, drivers and infrastructure. Several efforts have been undertaken to replace the DSM with message passing protocols across stands-based networks [Bal05a], [Mut04b], [Jon05], but these do not change the fundamental shared-memory architecture at the core of SWEDAT itself. For example, the Shared Memory Interface Likability Engineering (SMILE) effort involved fundamental changes to both JIMM and SWEDAT in order to replace the DSM with message-passing. As a JIMM modification, this effort promoted automatic conversion of indices (given data offsets) referencing SWEDAT data to pointers and pertinent automatic conversion of "endian" data (Figure 7). From the SWEDAT perspective, this effort promoted the transfer of data over a standard network via an object request broker.

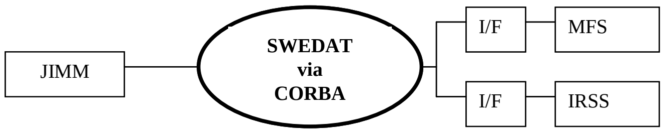


Figure 7 – Notional Exercise using JIMM and SMILE with Manned Flight and the IR stimulator.

The initial prototype that was built on IBM PCs and Linux was successful and as efficient as the legacy implementation [Bal05a]. Despite this success, the prototype implementation did not interoperate on all officially JIMM supported computer platforms and posed certain technical compatibility issues that kept it from being incorporated in the current JIMM distribution [Bal05b]. A similar effort funded by the Northrop Grumman Corporation (NGC) focused on employing SWEDAT via the Message Passing Interface (MPI) [Jon05]. MPI is a language-independent communications protocol common in parallel computing for interoperating between processes distributed over a network. The implementation was successful, though it was shown to have some time lags when compared against direct shared memory. Both of these efforts essentially extend the existing shared memory by adding a message passing protocol.

There is, however, a proposal for actually replacing the current shared memory protocol with a message-passing protocol. Between JIMM's semantic network known as the general array and the current shared memory implementation of SWEDAT is an internal dispatch system that coordinates instructions for changing array items and the sentence-like responses of the predicate logic coming from the model. Consequently, the dispatches currently employed within SWEDAT as well as common updates such as position and orientation of platforms could be encapsulated as messages and sent over a network directly and thus replace the SWEDAT protocol with a native message passing implementation. Furthermore, the implementation in JIMM could leverage JIMM's multi-threaded architecture [Mut04b]. Unfortunately, this has only been proposed and not actually implemented, although recent work to improve accuracy of position and orientation for interfaced systems and reduce network traffic [Mut07] could facilitate this effort.

4 An Alternative Perspective

An alternative approach is not simply a shared memory replacement addressing ACETEF data interchange, because it must address the end-to-end concurrency of physical battlespace interactions, i.e. the net effect of the shared memory SWEDAT with the JIMM virtual world notion as previously discussed.

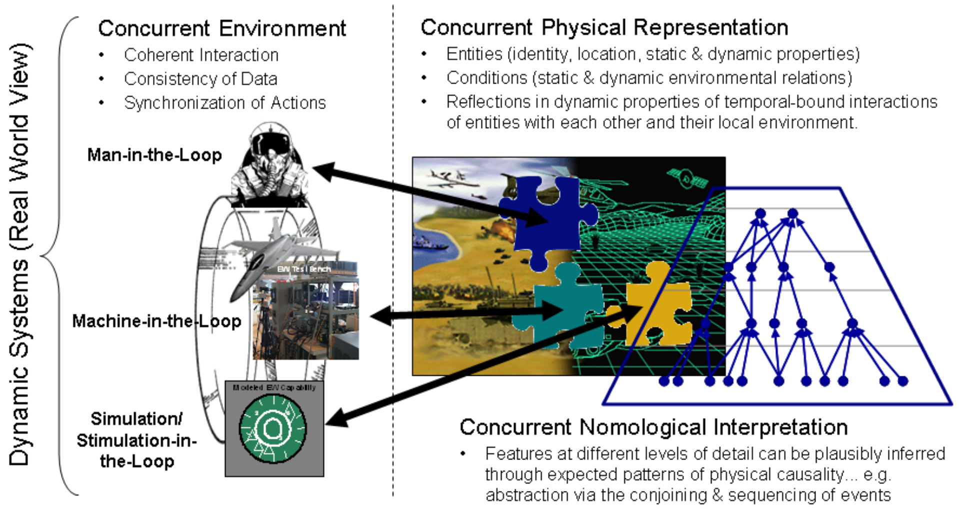

Dynamic Systems (Real World View)
Concurrent Environment
• Coherent Interaction
• Consistency of Data
• Synchronization of Actions
Man-in-the-Loop
Machine-in-the-Loop
Simulation/ Stimulation-in-the-Loop
Concurrent Physical Representation
• Entities (identity, location, static & dynamic properties)
• Conditions (static & dynamic environmental relations)
• Reflections in dynamic properties of temporal-bound interactions of entities with each other and their local environment.
Concurrent Nomological Interpretation
• Features at different levels of detail can be plausibly inferred through expected patterns of physical causality... e.g. abstraction via the conjoining & sequencing of events

Figure 8 – The road to an alternative perspective. The concurrent environment is reflected in a concurrent physical representation and a concurrent nomological interpretation for dynamic systems representing a "real world."

Consider that the ACETEF "real world" consists of dynamic systems of different types that must concurrently interoperate, as in Figure 8. The concurrent environment is marked by coherence of interaction, data consistency and event synchronicity. Individually, each dynamic system could be viewed as being part of a larger concurrent physical representation. The physical intersect will have physical entities within a physical environment that are both definable by various static and dynamic properties. The entities exist in space and thus have physical location as a common (often dynamic) property. The temporal-bounded interactions of entities with each other and with their local environment will be reflected in the time-evolution of the affected dynamic properties. The level of detail of the dynamic systems will typically differ as each provides its own functionally-oriented perspective of the physical representation. These differing levels of detail are not independent, but rather are abstractions of one another with respect to underlying cause and effect relations, i.e. physical laws. Within the context of time and space, this nomological relatedness manifests itself in anticipatory patterns of conjoined and sequenced events so that a plausible inference exists for concurrently interpreting features in the physical representation between levels of abstraction [Hum55]. This concurrent nomological interpretation in conjunction with the concurrent physical representation provides a basis for mediating concurrency between the dynamic systems through a common physical intersect.

A first opportunity to explore this concept presented itself in a 2005 project sponsored to take a given tactical situation and compare the effects of employing two different communications paradigms across certain mission profiles. The comparison reduced to two salient, operationally significant questions: (1) Does the "to-be" communications paradigm shorten the kill chain as compared to the "as-is" paradigm? (2) Does the "to-be" communications paradigm improve the Common Operational Tactical Picture (COTP) over that of the "as-is" paradigm? The desire was to "see" the data products as runtime visualizations of the kill-chain formation (logical graphs) and the tactical picture degree of commonality (statistical charts).

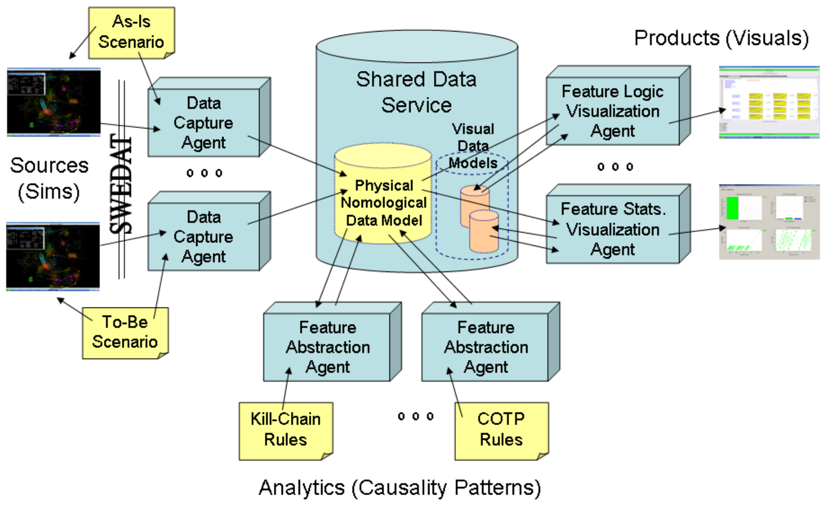

As-Is Scenario
Sources (Sims)
SWEDAT
Data Capture Agent
Data Capture Agent
To-Be Scenario
Shared Data Service
Physical Nomological Data Model
Visual Data Models
Products (Visuals)
Feature Logic Visualization Agent
Feature Stats. Visualization Agent
Feature Abstraction Agent
Feature Abstraction Agent
Kill-Chain Rules
COTP Rules
Analytics (Causality Patterns)

Figure 9 – Distributed agents concurrently sharing data services centered on a nomological data schema.

The challenge was to concurrently capture data from multiple sources; analytically processes the data for different causal features in parallel and map select identified features to multiple visual presentations on-the-fly. As shown in Figure 9, a concurrent TCP/IP based runtime environment centered on shared data services and a physical, nomological data schema was created along with agents for data capture, feature abstraction, and feature visualization. In this case, JIMM simulations were used to model the "as-is" and "to-be" situations. Consequently, two capture agents were employed to map both *a priori* scenario data and runtime event data into the data and metadata structure of the nomological schema for the synchronized simulations. The one side of the capture agent is necessarily specific to the data source, while the other side largely depends on the more generalized nomological data structure services. The physical features relevant for assessing kill-chain and COTP are at a higher level of abstraction. A generalized, model-driven causality correlation core was constructed so that identical feature abstraction agents could be deployed with different causality patterns. Consequently, two agents were used in this case, one for abstracting kill-chain related features and one for abstracting COTP-related features. The mode of data visualization determines the data structure, e.g. a scengraph for scene rendering, tables for chart data, and node-edge graphs for logical data representations. In this case, two agents corresponding to chart presentation and logical graph presentation were created. While the one side of each agent facilitated selecting features for presentation and mapping the features from the nomological structure to the particular data visualization structure, the other side of the agent mediated the data passing to a particular rendering library.

Drawing on the success of this initial effort [LB06], an even more generalized, higher performance system for integrated analytical purposes is currently under development. This current effort involves several noteworthy architectural features. First, a very flexible and efficient concurrent environment is created using the Tool Command Language (Tcl). Tcl is a growing language that was born in the early 1990's as a functional programming extension to the declarative C/C++ programming language. In functional programming languages one describes what to do in an imperative environment as opposed to how to do it in a declarative environment. Like its more famous functional programming ancestor LISP, Tcl is list-centric. In fact, Tcl has highly evolved lists, array and string processing capabilities including regular expressions and substitutions. Intended as an extension of C, these core capabilities are highly optimized for near byte-code level efficiencies. Furthermore, Tcl is remarkably platform-independent and can be either natively embedded within a C application or run independently with the ability to call linked libraries. Automatic memory management and a lack of pointers make Tcl very robust and highly compatible when embedded.

The Remote Procedure Call (RPC) script is a mere few-dozen lines of optimized code that allows command strings to be executed on a remote interpreter with the completion code and result being returned locally in the same way as if the command string had been executed locally (Figure 10).

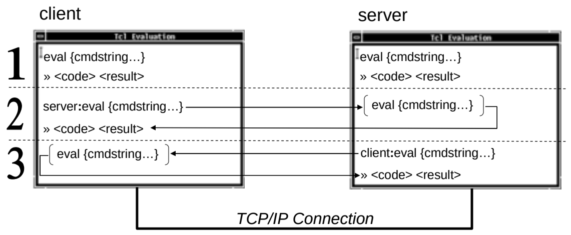


Figure 10 – Remote Procedure Call (RPC) service running on Tcl interpreters across TCP/IP provides an efficient basis for higher-order concurrent processing services.

A second architectural feature is the use of a modern In-Memory Database System (IMDS). The IMDS is the general-purpose descendant of the embedded database that take advantage of the large memory and very fast CPUs of today's computing platforms. In most respects, IMDS operate like embedded databases in that they are typically an integral part of the application they serve so that the database code is executed only when invoked by the application. The database's code may be in-line with the application code or called through linked libraries. In particular the Metakit database is currently being explored because it can be natively embedded in both C code and Tcl interpreters. In fact, the Tclkit release of Tcl uses Metakit internally as the real-time virtual file system. The Metakit database also employs a reduced, yet highly efficient instructional set that avoids the overhead of a full query language while still maintaining the necessary data consistency and flexibility of searching, sorting and selecting expected of a database engine. Unlike traditional databases that operate through transactions to the database file, usually through some caching mechanism that is all part in parcel of its data consistency model, the IMDS operates on a more direct data-in-memory model. Consequently, it typically requires fewer, simpler processes by eliminating or greatly simplifying the concepts of caching, data transfer, transaction processing, synchronization and rollback without loss of consistency. Furthermore, instead of a transaction-oriented data file, a Metakit repository can operate purely in-memory or be linked to file that is a more direct reflection of the data-in-memory, which makes operations like commits and loads tremendously fast and efficient.

When combined with the RPC service, the IMDS allows the creation of a reflective shared embedded database service with a few-dozen additional lines of functional programming code. In particular, the server establishes an in-memory database for each database service that it hosts. Each database at the server-side is linked to a file so that persistence is ensured. When a client connects and invokes the database service, its own embedded database is started in a pure in-memory mode and a data load of the data in the server's database for that service is downloaded. In actuality, the remote call service at this point takes special advantage of the fact that its communications exist in a slave interpreter as well as the ability to define code on-the-fly in an interpreter environment. In particular, the server uses the remote call service to set up another, special binary connection using the same physical connection for passing and loading the database data and structure on the client in low-level, binary form with extreme efficiency. It then destroys the special connection on both sides when the loading is complete without impacting the remote call service.

An application in which the client interpreter is embedded can now make database calls for creating, reading, updating and deleting both data and structure in addition to the efficient searching, sorting and selecting that databases are known for. If the call does not change the data or structure, then the call is executed by the embedded database locally. However if the call would result in a data or structure change, then the underlying remote call service, upon which the database service is constructed, passes the database command with any arguments to the server where it is first executed by the server's embedded database. The server then uses the same underlying remote call service to remotely execute the database command on all of the subscribing clients' embedded databases. By design, the server remotely executes the database command on the originating client's embedded database last. In this very simple way, the client knows that when it sees the change, then everyone already sees the same change!

Additionally, the database service provides a special connect/disconnect procedure for the underlying remote call service to invoke on connect/disconnect events as part of the overall coherency and consistency control at the database service level. When the database's own consistency model is combined with this simple yet effective coherency protocol, the resulting concurrency model now addresses the atomicity of data-in-structure as opposed to mere data. Since the database schema in this case reflects the physical features and nomological interpretation basis, the effective concurrency model can be managed at the appropriate battlespace interaction level.

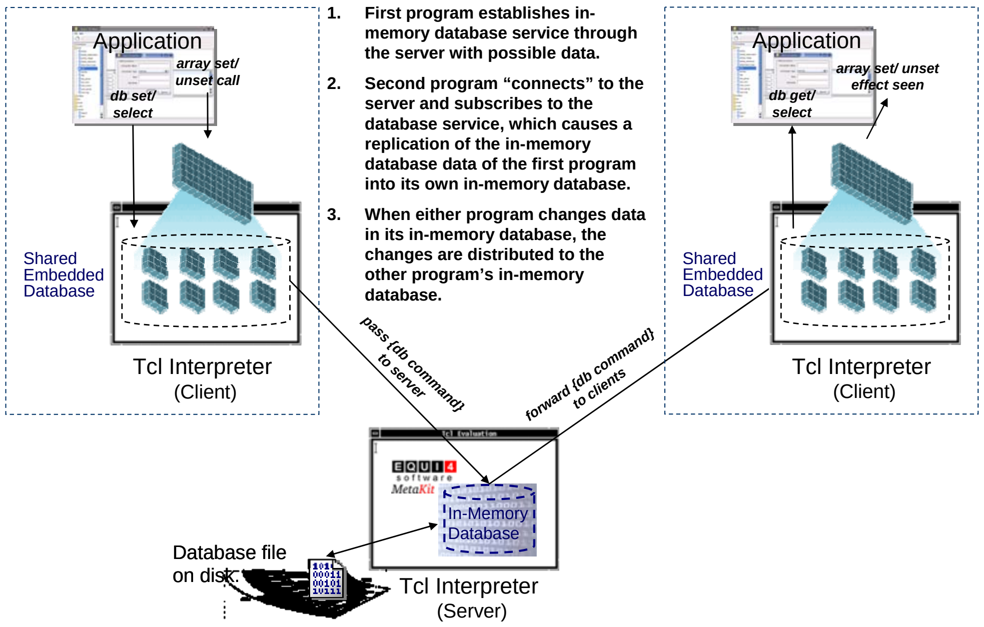
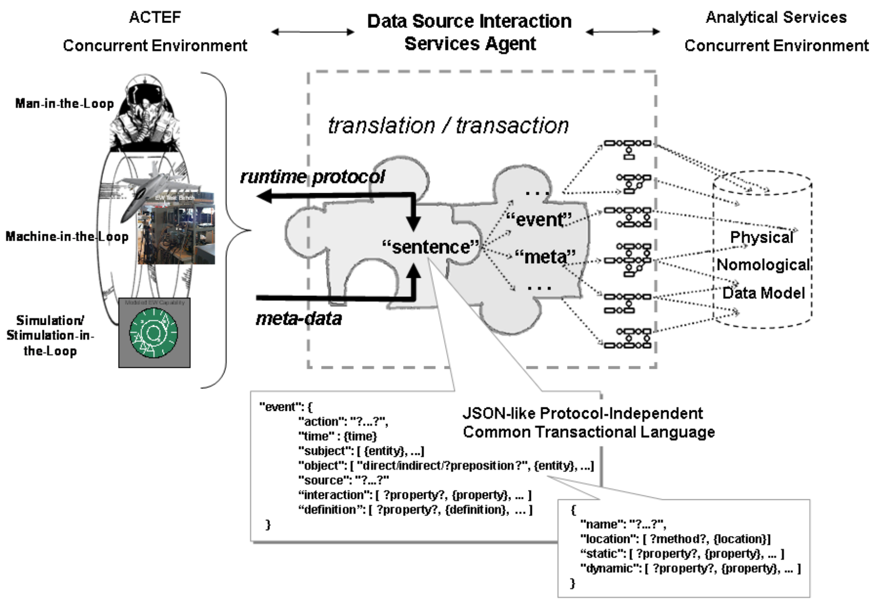

Application
array set/ unset call
db set/ select
Shared Embedded Database
Tcl Interpreter (Client)
1. First program establishes in-memory database service through the server with possible data.
2. Second program "connects" to the server and subscribes to the database service, which causes a replication of the in-memory database data of the first program into its own in-memory database.
3. When either program changes data in its in-memory database, the changes are distributed to the other program's in-memory database.
Application
array set/ unset effect seen
db get/ select
Shared Embedded Database
Tcl Interpreter (Client)
pass {db command} to server
forward {db command} to clients
EQUI4 software MetaKit
In-Memory Database
Database file on disk
Tcl Interpreter (Server)

Figure 11 – Reflective, shared in-memory database system services embedded within the Tcl interpreter and running on top of the RPC service enables concurrent sharing of data within the context of its database structure.

When the interpreter is embedded within a compiled application, a shared array variable between the interpreter and the compiled program can be a very effective data sharing metaphor. As an interpreter, Tcl uses late-binding so that variables are untyped and thus easy to share. It also has a full runtime state engine with event loops that can be used efficiently to trace variable manipulations and procedure executions. In this case, runtime variable traces allow the application to directly set/unset array elements while automatically propagating these changes to the underlying database service, which in turn uses the even deeper-seated remote call service to reflect any changes to the server and beyond. This overall reflective, embedded database concept is depicted in Figure 11.

On top of the database services are constructed the nomological abstraction services that apply patterns of conjoined events and event sequences to identify instances that translate into events and corresponding physical features at appropriate levels of abstraction. For such linguistic-inspired translations, the highly efficient internal list, array and string processing capabilities of the runtime Tcl environment prove advantageous. Furthermore, since all the services are modular, the resulting system becomes a service-of-services construct. For example, the database services act essentially as though all data is local. The underlying RPC services handles where the call is actually executed.

Finally, the essential nomological and physical feature database queries can be generalized if there were a protocol-independent common transactional language as an intermediate language for passing both data and meta-data in and out of the physical, nomological database (Figure 12).



```
"event": {
        "action": "?...?",
        "time" : {time}
        "subject": [ {entity}, ...]
        "object": [ "direct/indirect/?preposition?", {entity}, ...]
        "source": "?...?"
        "interaction": [ ?property?, {property}, ... ]
        "definition": [ ?property?, {definition}, ... ]
 }
```

```
{
  "name": "?...?",
  "location": [ ?method?, {location}]
  "static": [ ?property?, {property}, ... ]
  "dynamic": [ ?property?, {property}, ... ]
}
```

Figure 12 – The source-specific runtime and meta-data translation and nomological database transaction components of the data source interaction agent currently in development.

This intermediate language should be lightweight, but able to represent rich-linguistic structure; it should be easily machine parsed by C-family languages yet still have a human readable form; it should be readily adaptable to the informational content of a broad range of runtime/dynamic systems data protocols; and it must be able to handle both runtime event information as well as nomological meta-data information. In this perspective, JSON (JavaScript Object Notation) makes for a good data interchange language candidate currently under investigation. JSON is a standard text format that is completely language independent but uses conventions that are familiar to the C-family of languages. A particular challenge with the data source interaction agent stems from the fact that many data protocols only address or stress the runtime event/state data, which leaves the meta-data an unresolved source-specific issue.

Should this new system prove equally successful, the expansion to provide full bi-directional concurrent operation of dynamic systems can then be explored. If successful, this might eventually offer an alternative to the SWEDAT-JIMM architecture currently used to concurrently integrate ACETEF activities.

5 The Bigger Picture

Service-Oriented Architecture (SOA) is an architectural style that treats business functionality as modular, interacting services in an on-demand environment. Because the services exist in this imperative environment, SOA adds yet another dimension to understanding concurrency. The ACETEF claimants individually represent business activities.

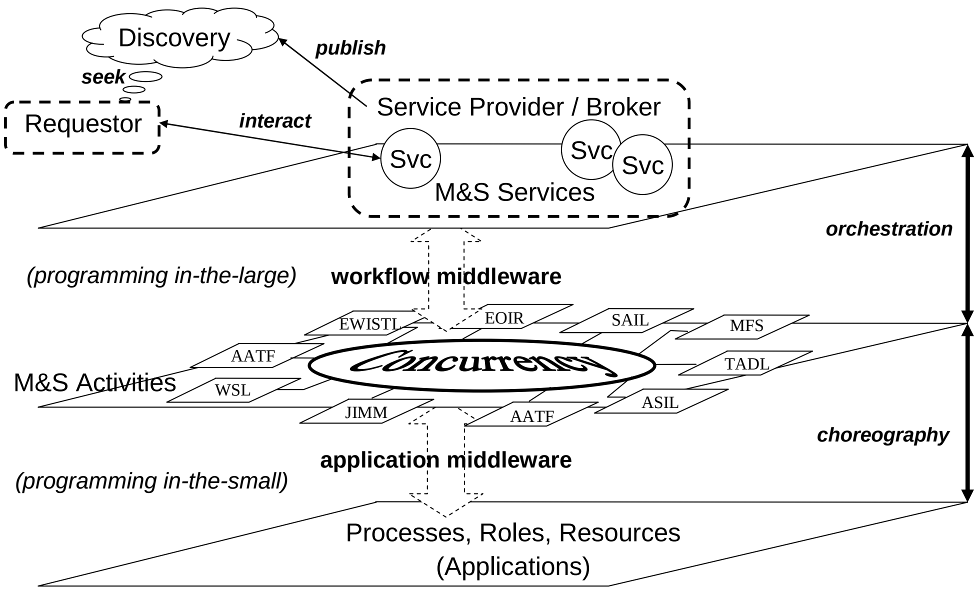


Figure 13 – A generalized SOA perspective of ACTEF.

These activities are actualized by the concurrent applications that interact through the sharing of processes, resources and roles. The concurrency aspect discussed thus far with both the SWEDAT/JIMM and the alternative RPC/nomological database approach represent application-level middleware. This is the lower side of the concurrent environment where the application interactions are “choreographed.” On the higher side of the concurrent environment is where the specific configuration of activities represented by the applications are “orchestrated” to form customer-level M&S based T&E services. This gives rise to the concept of workflow-level middleware of the concurrent environment. Figure 13 shows these two middleware perspectives. In an overly simplistic manner, this is similar to the distinction between simulation control and facility executive. Both perspectives involve the configuration, execution and synchronization of processes within the concurrent environment. These two perspectives were originally described as programming in-the-small and programming in-the-large respectively [DRK76], and today provide the supporting architecture beneath the service broker, requestor and discovery notions of the SOA business design pattern and distributed business approach.

The bigger SOA picture of ACETEF implies that the application-level middleware of the concurrent environment, which has been the primary focus of this paper, be combined with the workflow-level middleware of ACETEF’s activities. With products such as Starship, TENA is attempting to address these programming-in-the-large services. Within ACETEF, the locally developed ARIES facility executive provides complementary services at this level as well. ARIES is actually an acronym meaning Automated Resources Initialization, Execution & Synchronization, which describes its major functions. It was designed around the facility configuration and executive management functions necessary to support integrated testing processes, analogous to the I&GTC specification of the mid-1990’s [IGT96]. Similar to Window-NT’s Hardware Abstraction

Layer (HAL), ARIES created a kind of Systems Abstraction Layer (SAL) by leveraging three basic concepts:

- Network of Workstations (NOW) Concept - A computer is a computer is a computer.
- Macro Parallel Virtual Machine (MPVM) Concept - A process is a software-based, I/O-bound activity involving initialization, execution and synchronization.
- Hardware, software, arguments and files can be co-managed as dependencies in support of a process concept and create corresponding services – correspondence with the major I&GTC functions.

The business argument behind the ARIES concept is rooted in the concept of capabilities management whereby the infrastructure can be commoditized so that greater investment of effort can be placed with the activities most directly related to the business product development – inversion of the IT investment pyramid (Figure 14) [LB04]. The ARIES executive was actually used in the delivered version of the comparative analysis system that was described in the previous section to give a “media player” look and feel for selecting TACSIT variations by clicking options in a matrix or a drop-down menu and running the system with simple start and stop buttons – as on a media player.

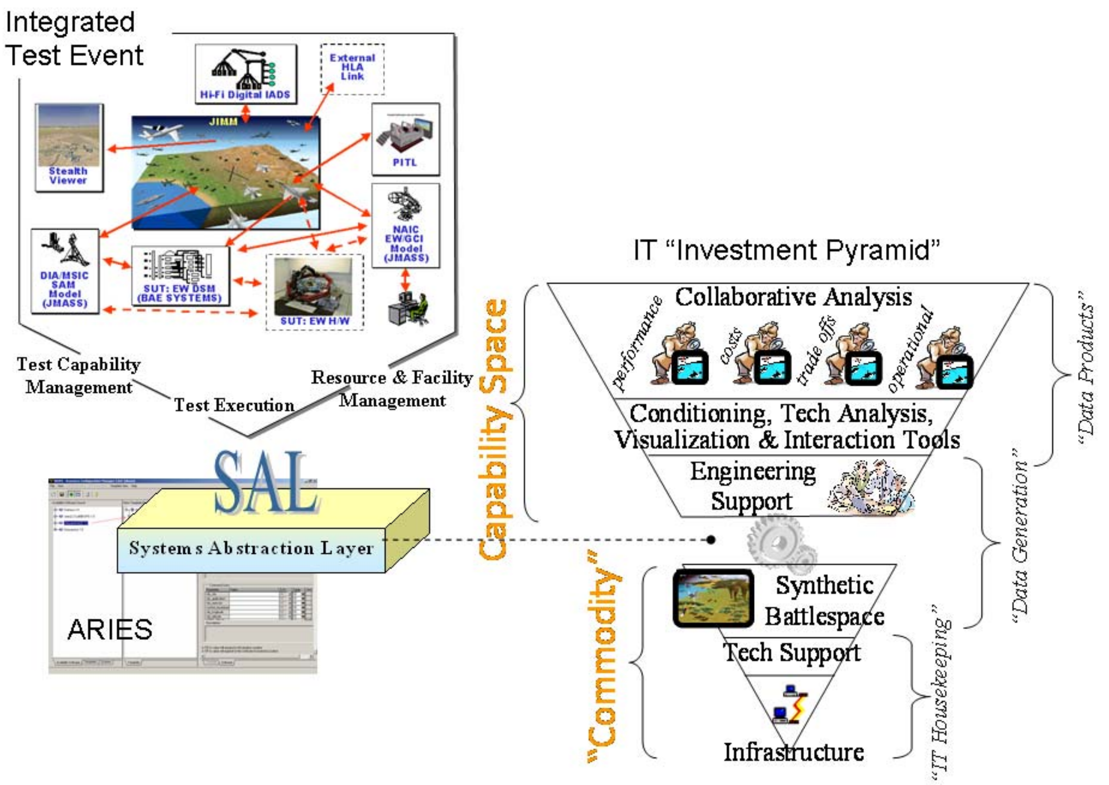


Figure 14 – The ARIES facility/test executive services for configuring and running integrated test events provides a systems abstraction layer that facilitates an inversion of the IT investment pyramid… greater investment at the customer data products level.

Some criticisms of SOA are based on the assumption that SOA is just another term for Web Services, which in turn implies the addition of XML parsing and composition overhead, for example. Certainly the concepts explored in this paper demonstrate otherwise. In fact, the SOA concept only implies a concurrent environment, of which the Web with the browser as a basic interpreter is a highly proliferated point in case. A concurrent environment created with interpreters and a basic RPC service is an alternative jumping-off point for creating an SOA, just as the alternative perspective described in the previous section using Tcl interpreters. Given the highly optimized and embeddable nature of the Tcl interpreters, this basic architecture with distributed procedure, file and databases services could possibly form a basis for other SOAs or even the programming in-the-large perspective of workflow-middleware. In any case, these are topics beyond the immediate scope of this paper deserving of possible further consideration and exploration.

6 Conclusions

When the operating paradigm is “Simulate-Stimulate-Analyze & Fix-Fly,” real-time, concurrent computing is going to be the order of the day. The ability to combine interoperability with concurrency both defines and distinguishes ACETEF and its stakeholders. As new systems are conceived to address modern warfighting doctrine, the need for applying this paradigm along the entire life-cycle spectrum from drawing board to operational support of these systems puts ACETEF’s capabilities in the spotlight.

ACETEF’s concurrency model is not merely about shared memory, but rather the combination of elemental entity data and event transactions, whose interchange fit a shared memory model reasonably efficiently, combined with the effective stimulus-decision-response interaction between the external world and reflected entities in a virtual JIMM world. While the learning curve for this approach can be steep, it has successfully supported environments with thousands of entities.

Since ACETEF’s original design and implementation, TENA has evolved as a CORBA-based protocol that is fundamentally able to address concurrency. Despite complaints of CORBA’s overhead, a recent Joint Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (JC4ISR) Interoperability Test and Evaluation Capability (InterTEC) exercise that ACETEF participated in demonstrated successful execution of environments with hundreds of entities. It is worth noting that the alternative approach suggested in this paper represents a degree of complexity that should be somewhere between the very simple shared memory and the very complex CORBA. As such, it is hoped that it shares the lightweight efficiency of the simple with sufficient flexibility of the complex to serve as a broadly applicable and viable middleware for concurrent application interaction. While this approach is currently focused on creating a common analytical environment for producing integrated data products reflecting the entire battlespace scenario, it may eventually be extendable to a full ACETEF middleware solution.

7 Acknowledgments

The authors would like to thank Jon Anderson, Stu Baldwin, Michael Chapman, Ross Jones, Leroy Mrozowski and Marc Briere for their contribution to this paper in the form of thoughts, reviews and comments.

References

[YM07] Young, Stu; Markowich, Amy; et al. "Integrated Battlespace Simulation & Test Strategic Plan". Integrated Battlespace Simulation and Test (IBST) Department (Code 5.4), Patuxent River, MD. 2007.

[OCD97] Air Combat Environment Test & Evaluation Facility Operations Concept Document, Simulation & Stimulation Division, Naval Air Warfare Center – Aircraft Division, May 1997.

[Ore02] Ören, T. *Growing Importance of Modelling and Simulation: Professional and Ethical Implications*, Invited Plenary Paper, Proceedings of the ICSC'2002 - The 5th Conference on System Simulation and Scientific Computing (Part of the Asian Simulation Conference).

[Alb03] Alberts, D. S., "Hayes, R. E. Power to the Edge Command... Control... in the Information Age," Command and Control Research Program (CCRP) Publication Series, www.dodccrp.org (2003).

[IEEE90] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[Ros97] Roscoe, A. W. (1997). The Theory and Practice of Concurrency. Prentice Hall. ISBN 0-13-674409-5.

[Lat95] Lattimore, Peter et al. "SWEG Users Guide". Air Combat Environment Test & Evaluation Facility (ACETEF), Patuxent River MD. 1995

[Lat07] Lattimore, Peter et al. "JIMM 3.0 Users Guide". JIMM Model Management Office (JMMO). Integrated Battlespace Simulation and Test (IBST) Department (Code 5.4), Patuxent River, MD. 2007

[LS07] Lohman, Gary E., Schaff, Josef, "*Back to the Future with JIMM*", JIMM Users Group Conference, June 2007, Solomons, Maryland. Available via the JIMM Model Management Office (JMMO).

[Mut05] Mutschler, David W. "Master / Client Feasibility Study". Available via the JIMM Model Management Office (JMMO), Integrated Battlespace Simulation and Test (IBST) Department (Code 5.4) at <jmmo@navy.mil>.

[IBST07] "Facility Overview Simulation and Stimulation". Integrated Battlespace Simulation & Test (IBST) Department (Code 5.4), Patuxent River, MD. 2007.

[MA06] Mrozowski, Leroy & Anderson, Jon et al. "ACETEF Plug-in Design Notes". Integrated Battlespace Simulation and Test (IBST) Department, Battlespace Modeling & Simulation Division (Code 5.4.2). Patuxent River, MD. 2006

[Mut03] Mutschler, David W. "The ACETEF Reflective Shared Memory Server v2". Air Combat Environment Test & Evaluation Facility". Patuxent River MD. 2003.

[Mut04a] Mutschler, David W. "Institute for HPC Real-Time Combat Environment Simulation Applications (RCESA)". Air Combat Environment Test & Evaluation Facility. Patuxent River MD. 2004.

[Mut04b] Mutschler, David W. "Thoughts on JIMM and Clusters". JIMM Users Group, Nov. 2004. Available via the JIMM Model Management Office (JMMO), Integrated Battlespace Simulation and Test (IBST) Department (Code 5.4) at <jmmo@navy.mil>.

[Bal05a] Baldwin, Stu. "Shared Memory Interface Likability Engineering (SMILE)". JIMM Users Group, Rosslyn VA, May 2005. Available via the JIMM Model Management Office (JMMO), Integrated Battlespace Simulation and Test (IBST) Department (Code 5.4) at <jmmo@navy.mil>.

[Bal05b] Baldwin, Stu. "Merging JIMMlib Including Feasibility over Internet". JIMM Users Group, Nov 2005. Available via the JIMM Model Management Office (JMMO), Integrated Battlespace Simulation and Test (IBST) Department (Code 5.4) at <jmmo@navy.mil>.

[Jon05] Jones, Ross E. "Application of Message Passing Interface (MPI) to JIMM". JIMM Users Group, Nov. 2005. Available via the JIMM Model Management Office (JMMO), Integrated Battlespace Simulation and Test (IBST) Department (Code 5.4) at <jmmo@navy.mil>.

[Mut07] Mutschler, David W. "Employing Path Information to Improve Accuracy in Distributed Simulations". Simulation Interoperability Workshop, Orlando FL, September 2007 (07F-SIW-033)

[Hum55] David Hume, "An Abstract of A Treatise of Human Nature", in An Inquiry Concerning Human Understanding, Bobbs-Merril, New York, 1955.

[LB06] Lohman, Gary & Briere, Mark. "Comparative Analysis in Runtime" JIMM Users Group, Sept. 2006. Available via the JIMM Model Management Office (JMMO), Integrated Battlespace Simulation and Test (IBST) Department (Code 5.4) at <jmmo@navy.mil>.

[DRK76] Frank DeRemer, Hans Kron, "Programming-in-the-Large Versus Programming-in-the-Small," IEEE Trans. on Soft. Eng. 2(2) 1976.

[IGT96] "Configuration, Data, and Facility Management Specification", Infrastructure and Generic Test Capability (I&GTC), Air Force Flight Test Center (AFFTC) at Edwards Air Force Base, Calif. 1996/97.

[LB04] Lohman, Gary E., Briere, Marc R. "*Architecting for T&E Capabilities Management*", Annual ITEA Symposium in conjunction with Army Test Week, Huntsville, Alabama (2004).

Biographies

GARY LOHMAN currently serves as Chief Scientist for AMEWAS Inc. Since 1993, Dr. Lohman has supported DoD as both civil servant and contractor across a myriad of information system, knowledge management, communications, modeling and simulation and warfare analysis projects. Prior to this, Dr. Lohman’s focus was research in massively parallel optical image processing, optical computing and holographic interferometric non-destructional testing. He was also involved in the design and construction of optical interconnection networks and the technical optical design of extremely long-baseline Fourier optical processing systems for the exploitation of both electro-optical spatial light modulators and non-linear optical crystals. He received his doctorate in theoretical and experimental physics with a minor in physics education from the Friedrich-Alexander Universität in Erlangen, Germany. While working in Germany, Dr. Lohman was part of the Modular Erweiterbare Multiprozessor-System (MEMSY) SFB-182 German supercomputing project and the EEC sponsored multi-national ESPRIT-II project New Architectures for Optical Processing in Industrial Applications (NAOPIA).

DAVID W. MUTSCHLER has worked as a computer engineer for the Naval Air Systems Command (NAVAIR) for twenty-two years. He has worked at ACETEF in support of JIMM and its predecessor SWEG for twelve of those years. He served as the JIMM Model Manager from June 2004 to February 2006. He obtained his Ph.D. in Computer and Information Science from Temple University in 1998 and is an Associate Professor in the Florida Institute of Technology University College.

Live • Virtual • Constructive

BATTLESPACE
modeling and simulation

NAVAIR

Dr.rer.nat. Gary E. Lohman

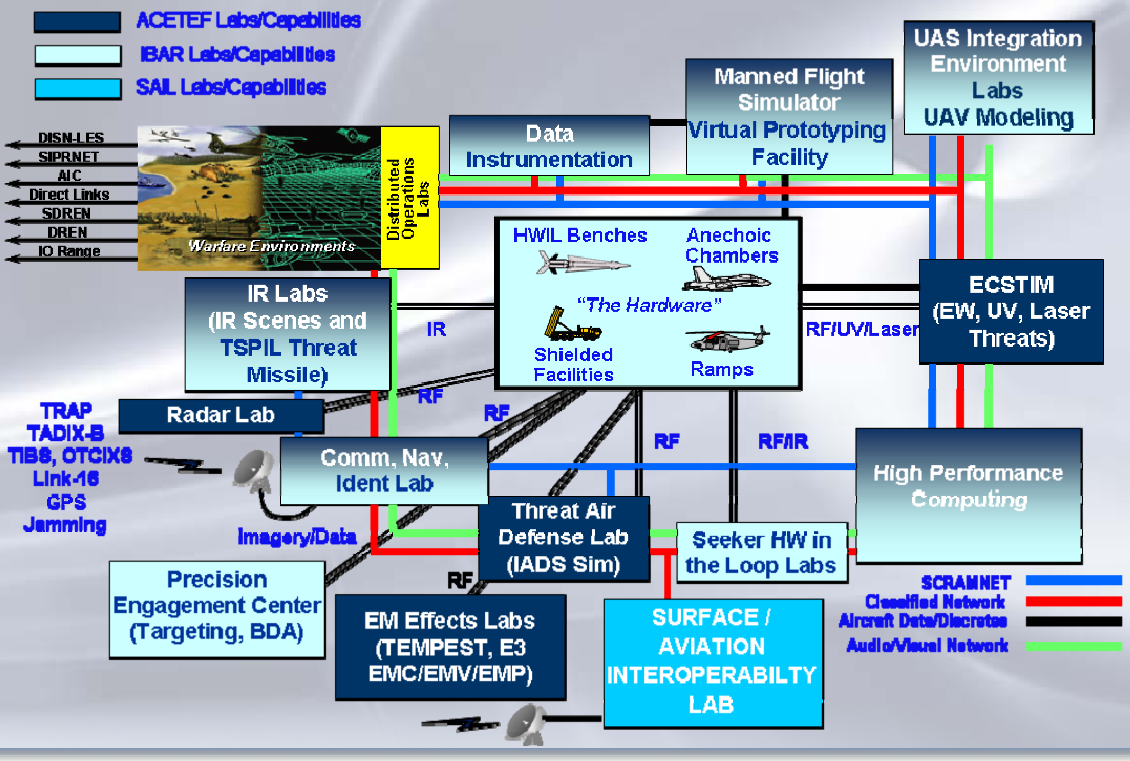
David W. Mutschler, Ph.D.

Modeling & Simulation in the T&E Environment
Las Cruces, New Mexico
Tuesday, 11 Dec 07

Shared Memory Architecture

and Explored Alternatives for Interoperability

- The Air Combat Environment Test & Evaluation Facility (ACETEF) is a major component of the Naval Air Systems Command (NAVAIR) Integrated Battlespace Simulation and Test (IBST) Department…

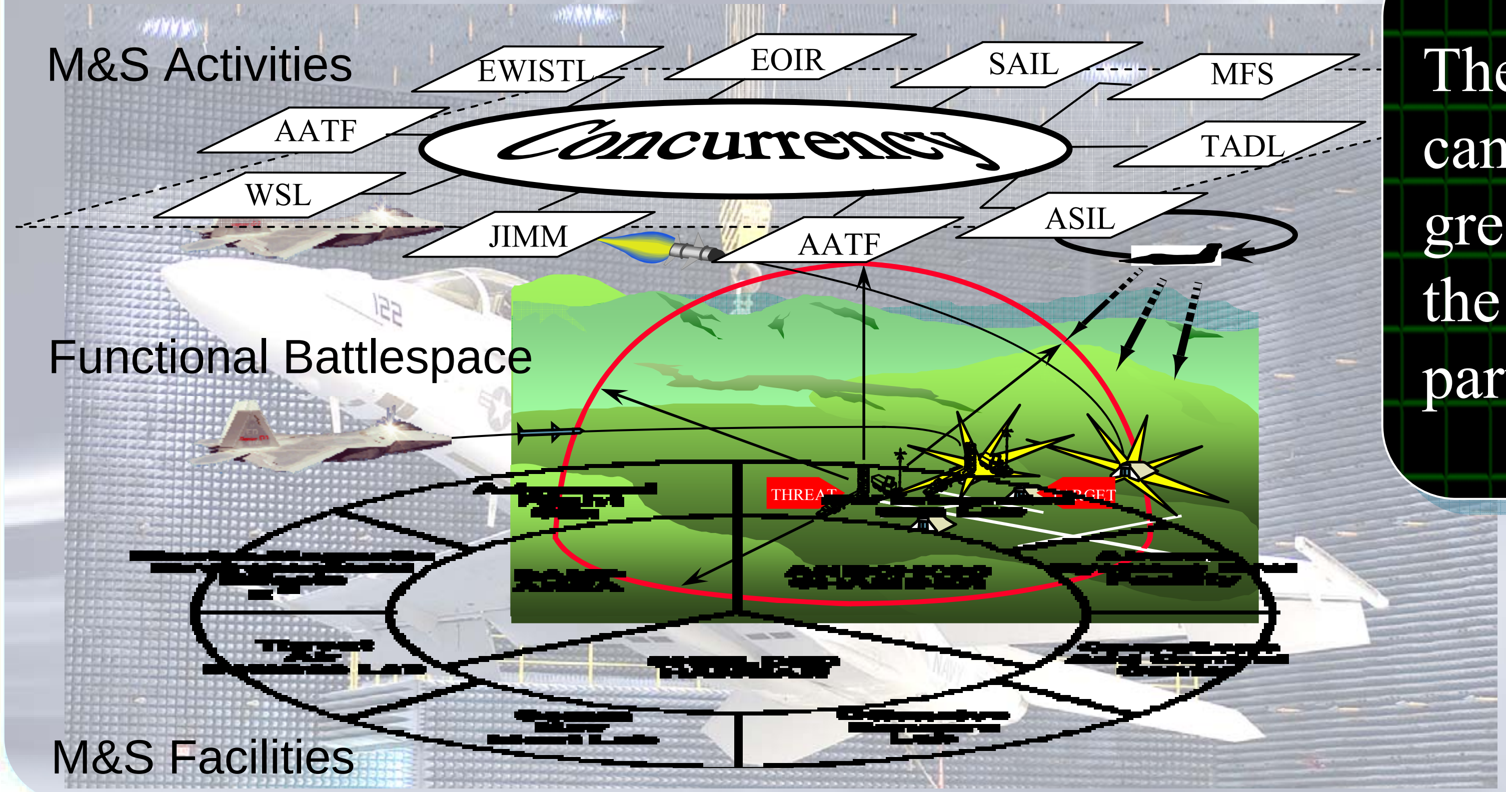


ACETEF Labs/Capabilities
IBAR Labs/Capabilities
SAL Labs/Capabilities
DISN-LES
SIPRNET
AIC
Direct Links
SDREN
DREN
IO Range
Warfare Environments
Distributed Operations Labs
Data Instrumentation
Manned Flight Simulator Virtual Prototyping Facility
UAS Integration Environment Labs UAV Modeling
HWIL Benches
Anechoic Chambers
"The Hardware"
Shielded Facilities
Ramps
ECSTIM (EW, UV, Laser Threats)
IR Labs (IR Scenes and TSPIL Threat Missile)
IR
RF
RF/UV/Laser
Radar Lab
TRAP
TADIX-B
TIBS, OTCIXS
Link-16
GPS
Jamming
Comm, Nav, Ident Lab
RF
RF/IR
High Performance Computing
Threat Air Defense Lab (IADS Sim)
Seeker HW in the Loop Labs
Imagery/Data
RF
Precision Engagement Center (Targeting, BDA)
EM Effects Labs (TEMPEST, E3 EMC/EMV/EMP)
SURFACE / AVIATION INTEROPERABILTY LAB
SCRAMNET
Classified Network
Aircraft Data/Discretes
Audio/Visual Network

Highly interconnected capabilities spanning the spectrum of ground testing for aircraft and aircraft systems.

The Organizational Architecture

- ACETEF's Paradigm - "Simulate-Stimulate-Analyze & Fix-Fly"
- ACETEF's Interoperability - Based on concurrency at a functional battlespace level.

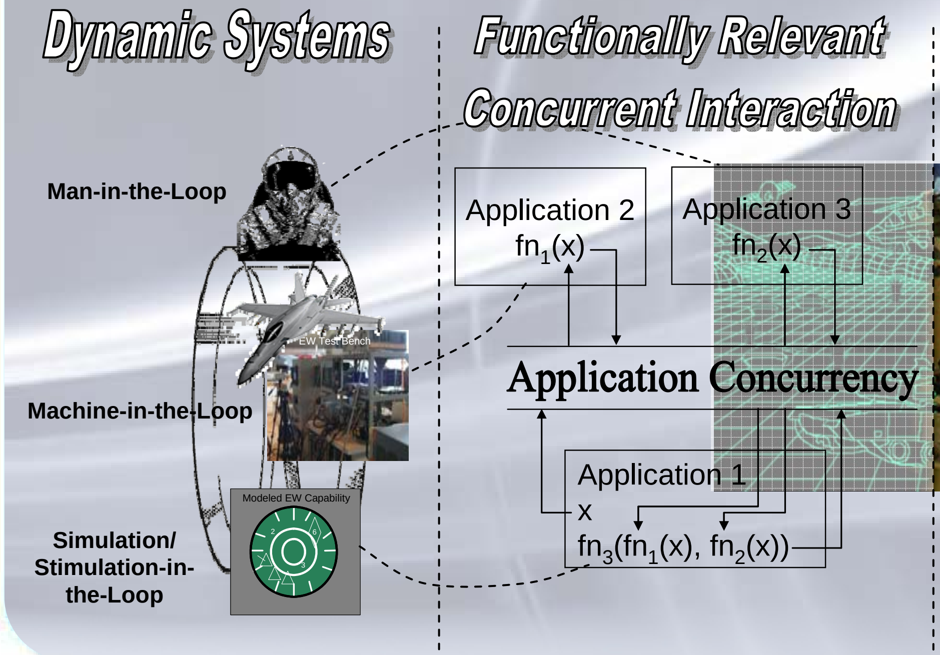


The whole can become greater than the sum of its parts!
"Π vs. Σ"

The ACETEF Concept

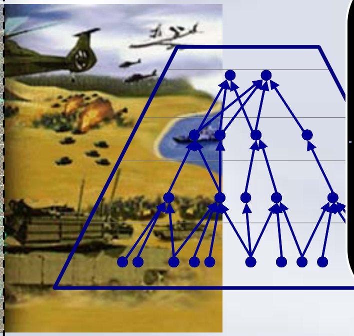
- Application concurrency implies some degree of…
  - Inter-Process Communication (Interconnectivity)
  - Coherence Protocol
  - Data Consistency Model
  - Synchronization Mechanism

"Coherent" Environment



Concurrent Physical Representation

- Entities (identity, location, static & dynamic properties)
- Conditions (static & dynamic environmental relations)
- Reflections in dynamic properties of temporal-bound interactions of entities with each other and their local environment.



Concurrency is a real challenge when the dynamic systems function at differing levels of abstraction!

Concurrent Nomological Interpretation

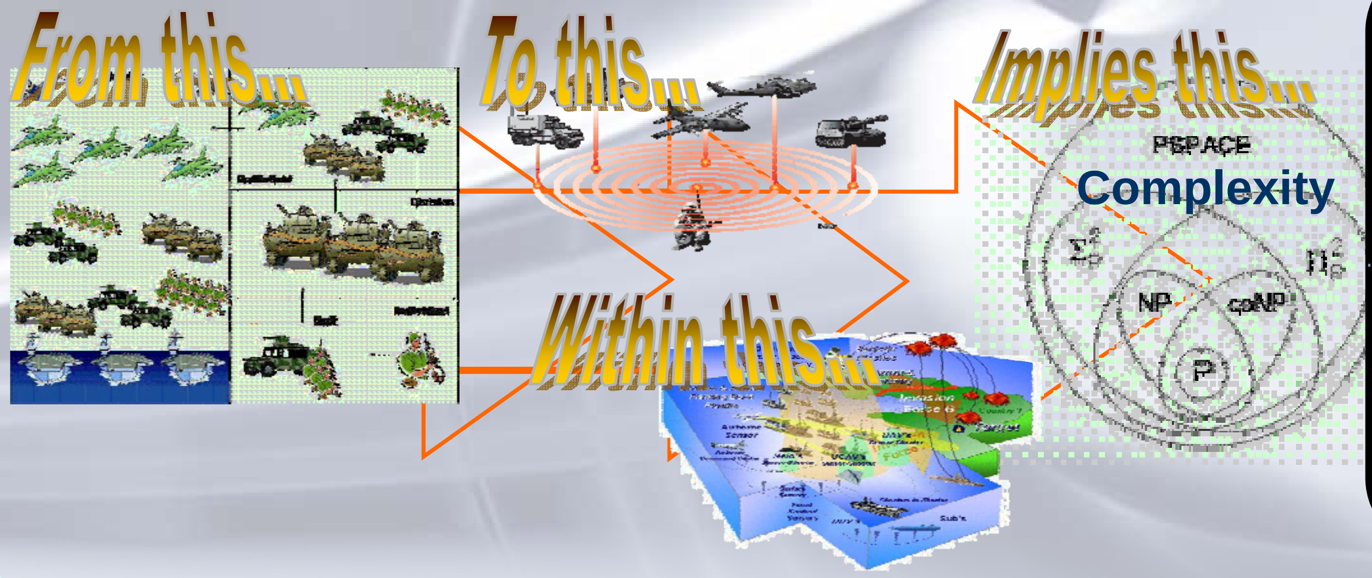
- Features at different levels of detail can be plausibly inferred through expected patterns of physical causality... e.g. abstraction via the conjoining & sequencing of events

Application Concurrency

- Historically in T&E, the degree of "coherence" could be greatly constrained by reducing the range of dynamic interactions and focusing more on the SUT's *reaction* vice full *interaction*.
- **BUT**… shift to dynamic control & re-planning OPS driven by multi-INT interactions facilitated by multi-mission / multi-INT / multi-interconnected platforms significantly ups this ante.

"more Σ than Π"

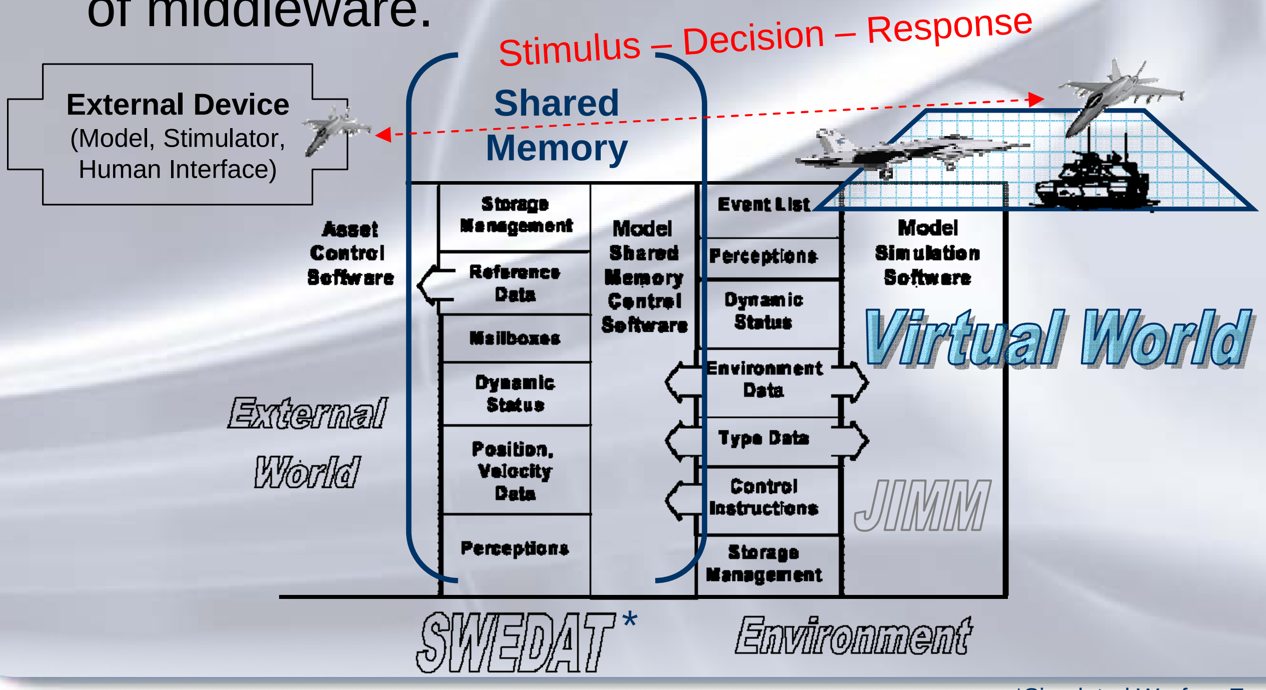
"more Π than Σ"



In a complex environment, there exists an opportunistic region conducive to adaptation between the multi-stable and the chaotic unstable regions… the *sweet spot*!

Situation » Required Degree of Concurrency

- Concurrency has meaning at the functional level - e.g. at battlespace interactions between the dynamic systems.
  - Real dynamic systems operate at different levels of detail.
  - Need nomologically consistent interpretation/transaction of data & events.
- ACETEF uses a semantic-model as a virtual-world, agent-based form of middleware.



Concurrency is not about the environment model, data protocol or shared memory per se, but rather their combined net SDR effect!

*Simulated Warfare Environment Data Transfer Protocol

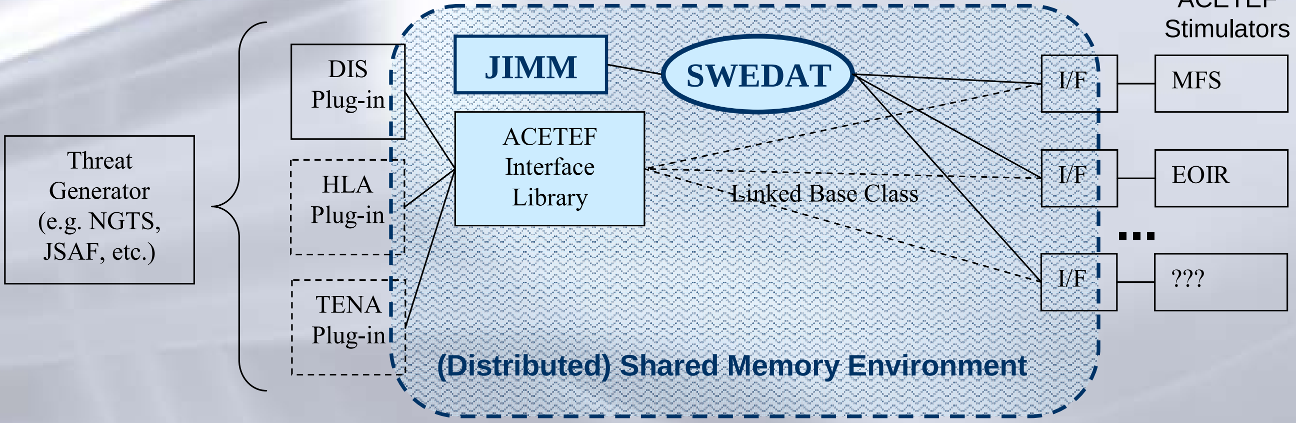
Traditional ACETEF Architecture

- Shared memory is both an inter-process communication (IPC) method and a simple concurrency model at the low level of variable-sharing » ***Two Basic Challenges***
  - Concurrency at the “variable-passing” level is not the same as battlespace functional level concurrency – interface logic can be non-trivial.
  - Distributed dynamic systems means distributed shared memory (DSM) infrastructure – can be costly and proprietary.

Current implementation employs:

- **An interface library with a linked base class to facilitate the necessary interface logic.**
- **A plug-in architecture to facilitate interfacing with DIS, HLA and TENA.**
- **Proprietary DSM infrastructure for direct interfacing.**

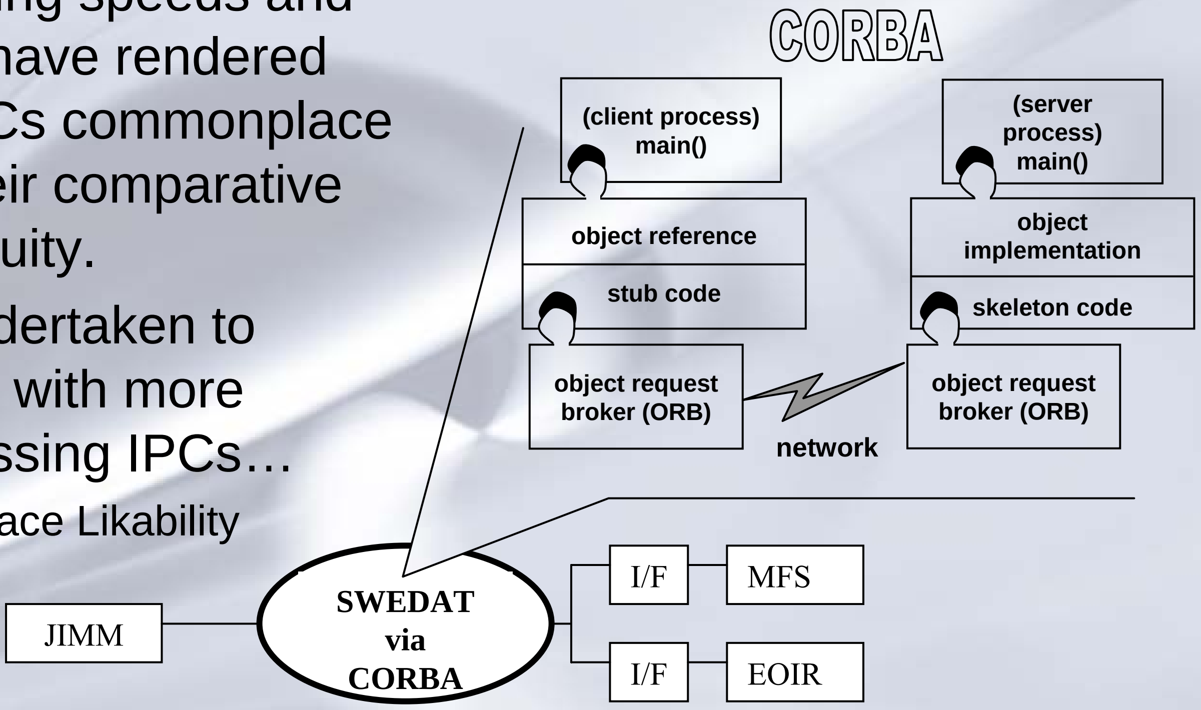
At ACETEF...



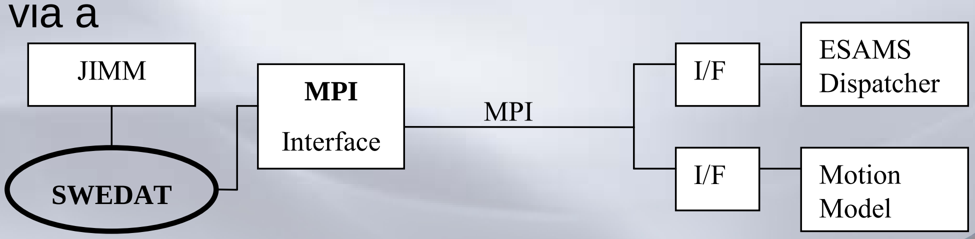
Shared Memory Architecture Challenges

Beyond the DSM Infrastructure…

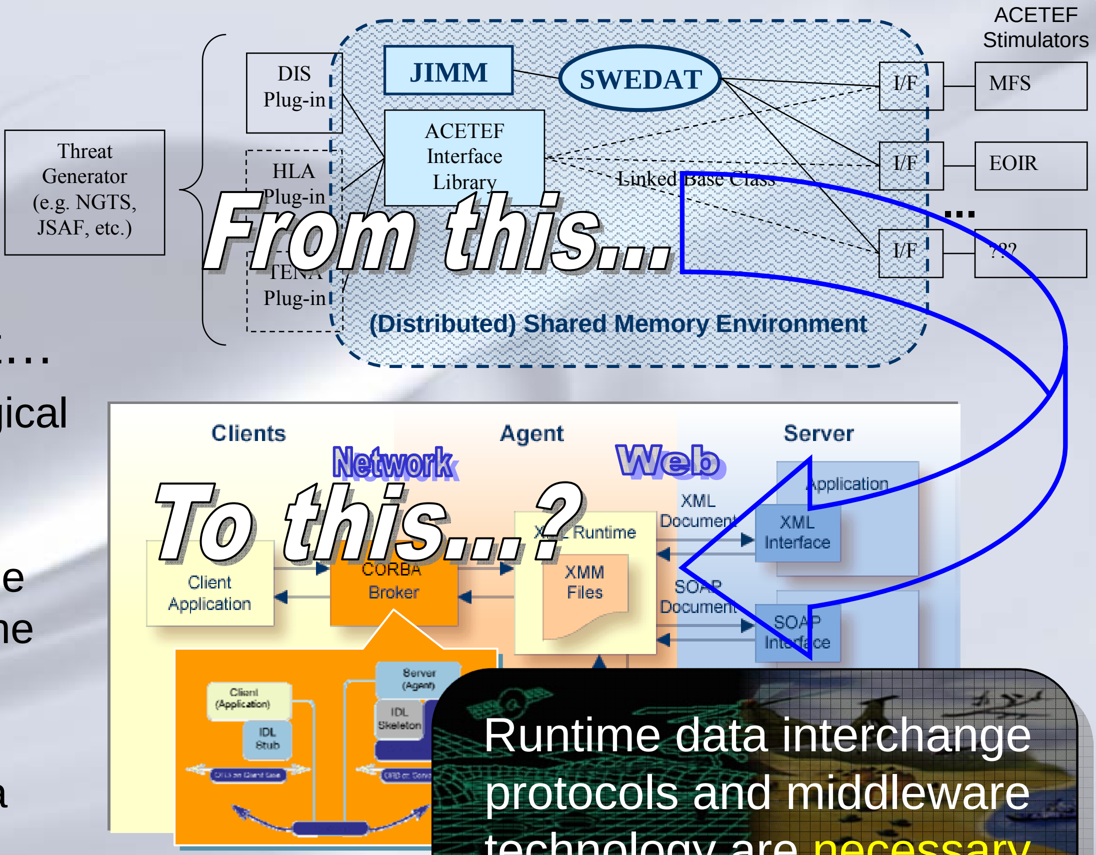
- Off-the-shelf processing speeds and network bandwidths have rendered message-passing IPCs commonplace – especially given their comparative ease-of-use and ubiquity.
- Efforts have been undertaken to replace the DSM IPC with more modern message-passing IPCs…
  - Shared Memory Interface Likability Engineering (SMILE)



  - Sponsor-funded effort - SWEDAT via a Message Passing Interface (MPI)



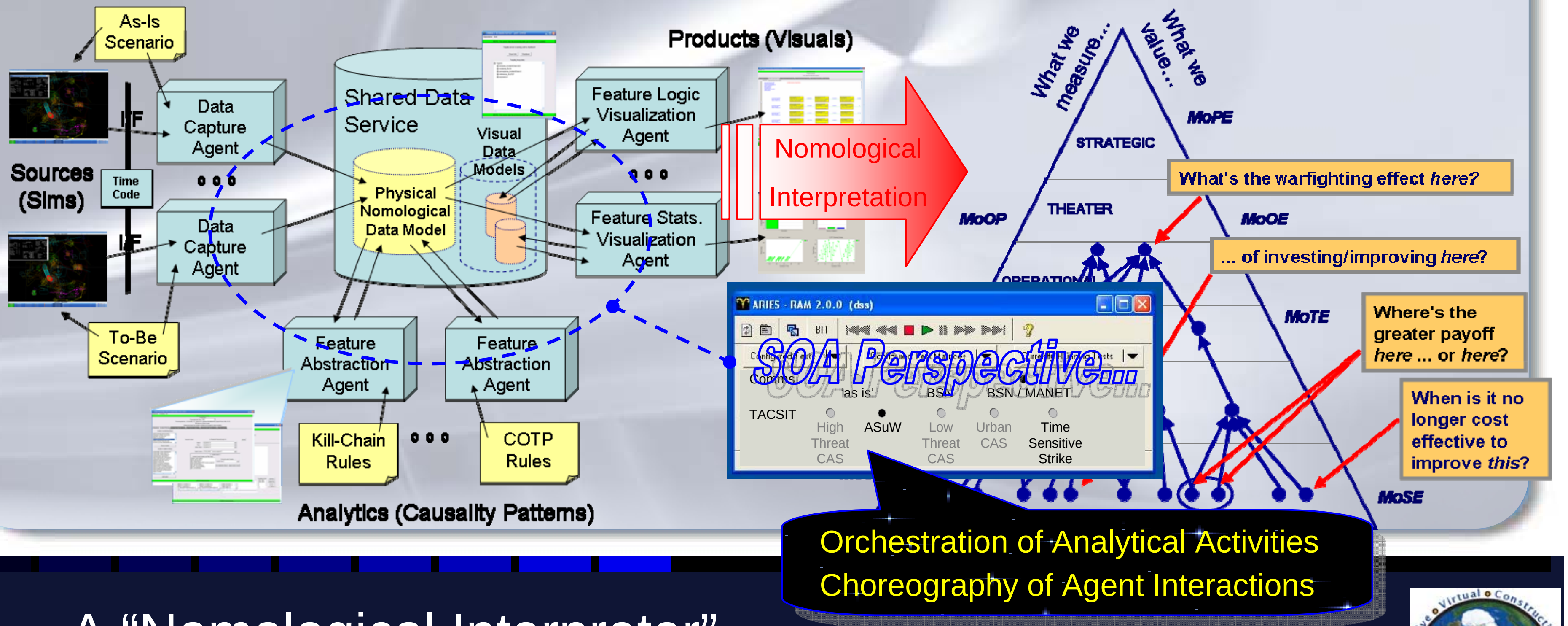
- The shared memory + SWEDAT + environment model co-determine ACETEF concurrency.
- A "new" architecture must…
  - Provide equivalent nomological interpreting between the dynamic systems.
  - Address the fact that runtime data & control does not alone provide the full nomological relatedness understanding.
  - Not exclude alternative data protocols & technical approaches – given the range of possible L-V-C dynamic systems.



Runtime data interchange protocols and middleware technology are necessary but not sufficient to ensure the desired concurrency.

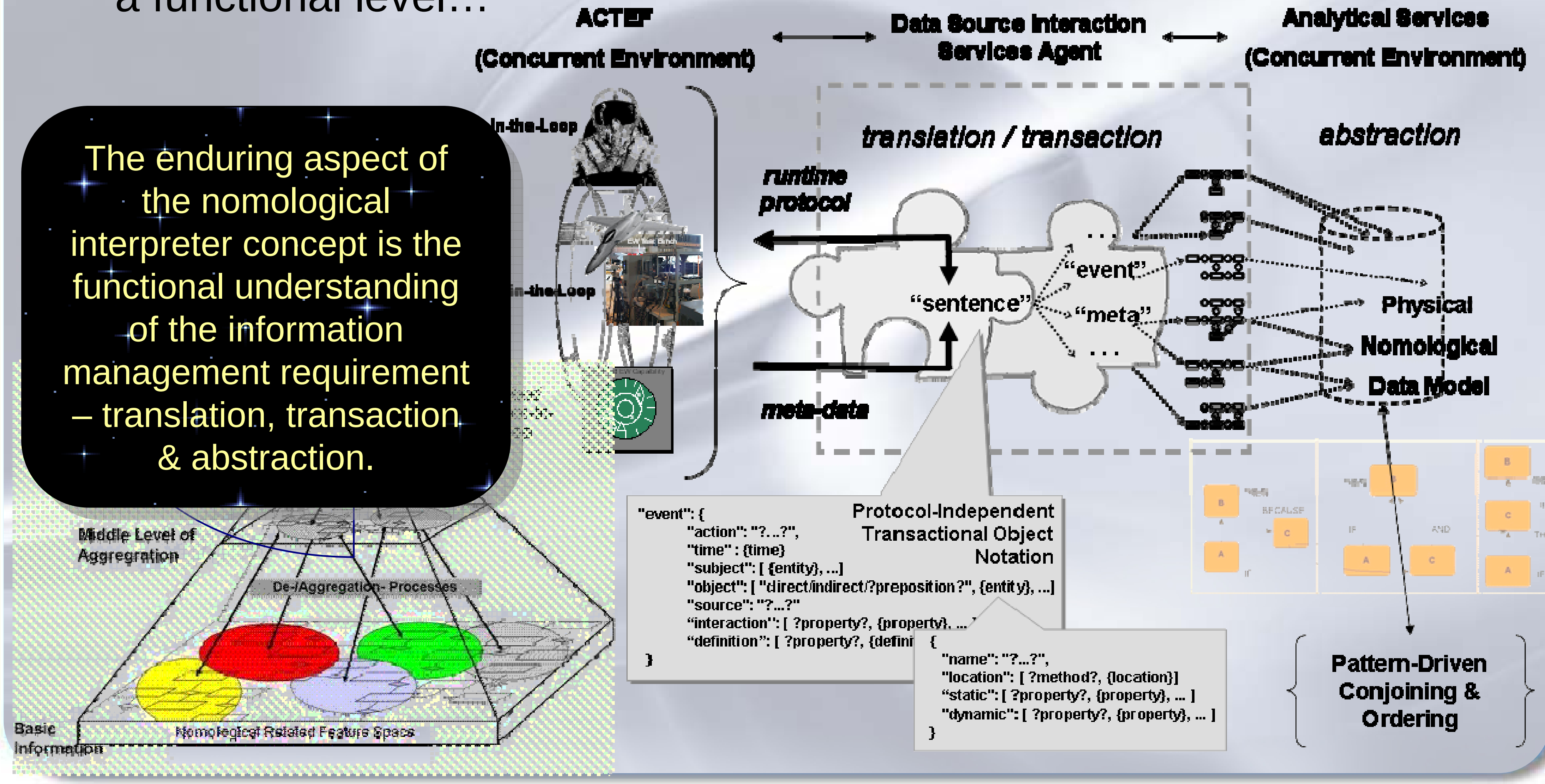
Beyond the Shared Memory Architecture

- Comparative analysis project illustrates potential alternative...
  - Required *runtime insight* of kill-chain evolution and commonality of the COTP – the *what & why* in addition to the basic *who and where*.
  - Runtime data + a priori metadata combined in a generalized physical representation of features within a nomological framework of relatedness.
  - TCP/IP, Agent-Based, rule-driven runtime abstraction of features provides interpreting between levels of abstraction… *building the visual data products*.


As-Is Scenario
Sources (Sims)
Time Code
I/F
I/F
To-Be Scenario
Data Capture Agent
Data Capture Agent
Shared Data Service
Physical Nomological Data Model
Visual Data Models
Products (Visuals)
Feature Logic Visualization Agent
Feature Stats. Visualization Agent
Feature Abstraction Agent
Feature Abstraction Agent
Kill-Chain Rules
COTP Rules
Analytics (Causality Patterns)
Nomological Interpretation
What we measure...
What we value...
MoPE
STRATEGIC
MoOP
THEATER
MoOE
OPERATIONAL
MoTE
MoSE
What's the warfighting effect here?
... of investing/improving here?
Where's the greater payoff here ... or here?
When is it no longer cost effective to improve this?
ARIES - RAM 2.0.0 (dss)
SOA Perspective...
Comms
'as is'
BSN
BSN / MANET
TACSIT
High Threat CAS
ASuW
Low Threat CAS
Urban CAS
Time Sensitive Strike
Orchestration of Analytical Activities
Choreography of Agent Interactions

A “Nomological Interpreter” Within a Services-Oriented Architecture

- Focus on solving the enduring information management challenges at a functional level…



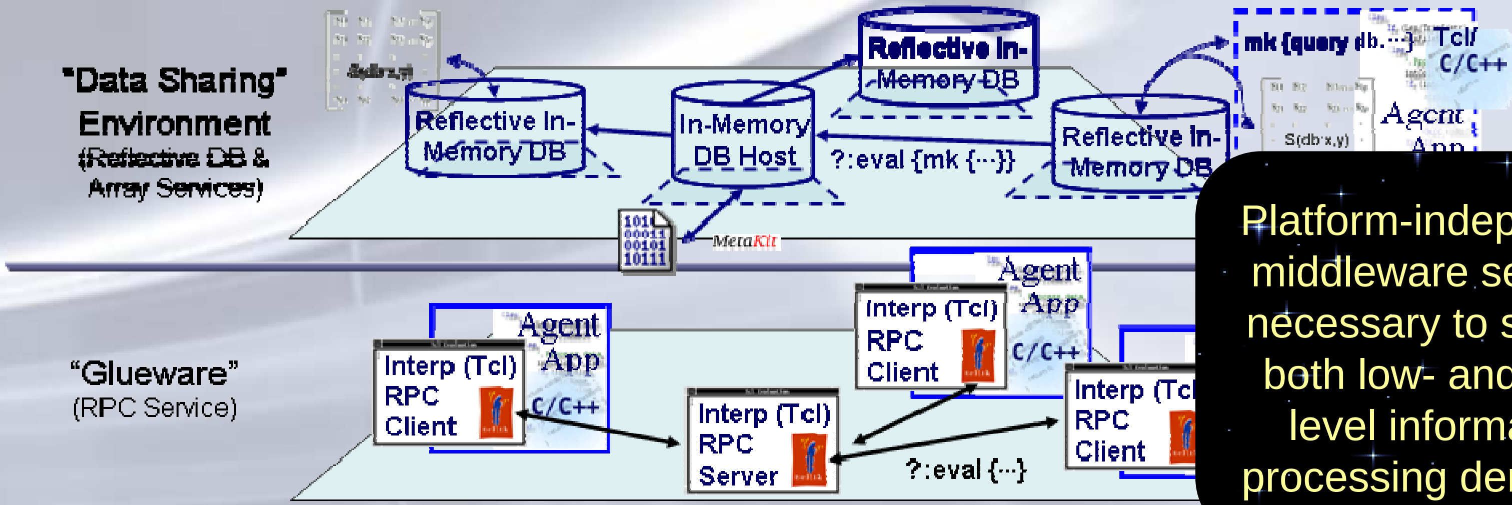
```
"event": {
        "action": "?...?",
        "time" : {time}
        "subject": [ {entity}, ...]
        "object": [ "direct/indirect/?preposition?", {entity}, ...]
        "source": "?...?"
        "interaction": [ ?property?, {property}, ...
        "definition": [ ?property?, {defini
 }
```

```
{
   "name": "?...?",
   "location": [ ?method?, {location}]
   "static": [ ?property?, {property}, ... ]
   "dynamic": [ ?property?, {property}, ... ]
}
```

The enduring aspect of the nomological interpreter concept is the functional understanding of the information management requirement – translation, transaction & abstraction.

The Conceptual Basis

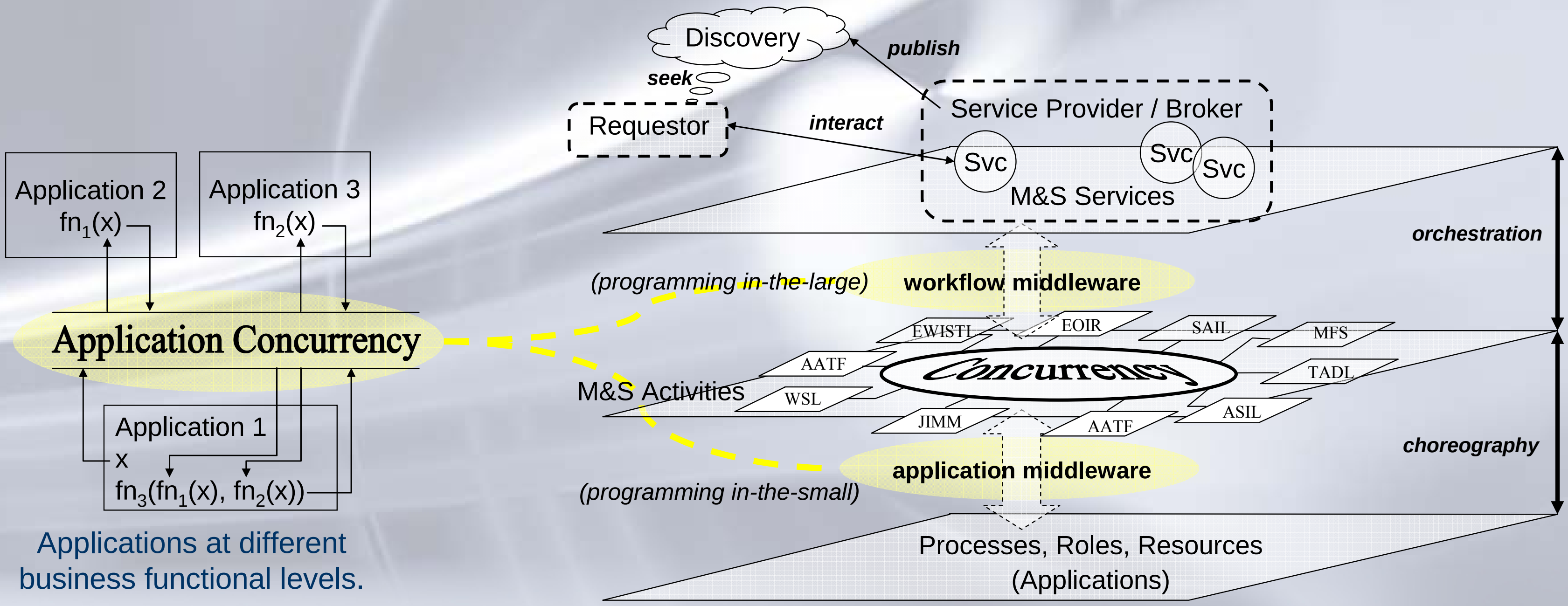
- RPC infrastructure across easily embedded, high-functioning byte-code compiling interpreters, e.g. efficient, general-purpose functional programming environments (list, array, string processing…).
- Distributed, interactive services… reflective variable sharing, reflective embedded database sharing, file sharing, efficient transactional object notation…



Platform-independent middleware services necessary to support both low- and high-level information processing demands.

The Technical Enabling Environment

- Business Services – the end-to-end capabilities perspective of discovering, requesting and producing relevant, useful M&S products.
- Orchestration Services – initialization, execution, synchronization & management of M&S activities.
- Choreography Services – concurrent integration of the dynamic systems and analytical tools behind the M&S activities.



Applications at different business functional levels.

Ability to Support a Larger SOA Perspective…

Conclusions

- These are exciting times!
  - The "Simulate-Stimulate-Analyze & Fix-Fly" operating paradigm is increasingly relevant with emerging systems and warfighting doctrine.
  - The "immersion in a coherent environment" concurrency model that defines and distinguishes ACETEF is a critical capability.
  - The on-going architectural endeavors to discover alternatives to the shared-memory based architecture at ACETEF are providing valuable opportunities…
    - Back-to-Basics look at the enduring information management challenges as related to dynamic systems, concurrency and physical causality.
    - Re-thinking of middleware concepts in light of emerging and maturing technologies as well as the larger SOA context.

"If you do what you've always done, you'll get what you've always gotten."

-anonymous